Program 1: Network Packet Handling

This is an INDIVIDUAL Assignment (Do not collaborate)

Instructions

Please read the entire instructions and the skeleton code provided for the server and the client before you start coding.

Setup

1. **Create two separate Python files** for the server and client, named `server.py` and `client.py`, respectively.
2. **Copy and paste the provided skeleton code** into the respective files.

Implementation

3. **Server**: Complete the server code by implementing the `unpack_packet` function. The server should accept connections, receive custom packets from clients, unpack the packets, and echo back the packet fields.
4. **Client**: Complete the client code by implementing the `create_packet` and `handle_packet` functions. The client should connect to the server, create packets based on user input and command-line arguments, and display the server's echoed responses.

Testing and Submission

5. **Test** your server and client by running them in separate VMs. Ensure that they can handle different types of payloads based on the header's service type.
6. **Error Handling**: Perform error handling, add comments for clarity, and optimize the code as needed.
7. **Submit** your completed `server.py` and `client.py` files for evaluation through the submission platform.

Server Skeleton Code

```
import socket
import struct

def unpack_packet(conn, header_format):
    # TODO: Implement header unpacking based on received bytes
    # TODO: Create a string from the header fields
    # return the string - this will be the payload
    return packet_header_as_string

if __name__ == '__main__':
    host = 'localhost'
    port = 12345

    # Fixed length header -> Version (1 byte), Header Length (1 byte),
Service Type (1 byte), Payload Length (2 bytes)
    header_format = ''  # TODO: Specify the header format using "struct"
```

```
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        conn, addr = s.accept()
        with conn:
            print(f"Connected by: {addr}")
            while True:
                try:
                    # TODO: Receive and unpack packet using the unpack_packet
function
                    payload_string = unpack_packet(conn, header_format)
                    pass
                except:
                    print("Connection closed or an error occurred")
                    break

             #TODO: create header

             #TODO: add payload

             #TODO: send to client
```

Client Skeleton Code

```
import argparse
import socket
import struct

def create_packet(version, header_length, service_type, payload):
    # TODO: Implement packet creation based on parameters
    # TODO: use the python struct module to create a fixed length header
    # TODO: Fixed length header -> Version (1 byte), Header Length (1 byte),
Service Type (1 byte), Payload Length (2 bytes)
    # TODO: payload -> variable length
    # TODO: depending on the service type, handle encoding of the different
types of  payload.
    # TODO: service_type 1 = payload is int, service_type 2 = payload is
float, service_type 3 = payload is string

    return packet

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Client for packet creation
and sending.")
    parser.add_argument('--version', type=int, required=True, help='Packet
version')
    parser.add_argument('--header_length', type=int, required=True,
help='Length of the packet header')
    parser.add_argument('--service_type', type=int, required=True,
help='Service type of the payload (1 for int, 2 for float, 3 for string)')
    parser.add_argument('--payload', type=str, required=True, help='Payload
to be packed into the packet')
    parser.add_argument('--host', type=str, default='localhost', help='Server
host')
    parser.add_argument('--port', type=int, default=12345, help='Server
port')

    args = parser.parse_args()
```

```
    # TODO: Create and send packet using the create_packet function
    packet = create_packet(args.version, args.header_length,
args.service_type, args.payload)

    #TODO: connect to the server

    #TODO: send the packet

    #TODO: recive the packet

    #TODO: prints header

    #TODO: prints payload
```

Assignment Rubric

Client Implementation (40 points)

- **Client Runs (10 points)**: The client code runs without errors.
- **Client Packs the Packet appropriately (10 points)**: The client can pack a packet.
- **Type handling**: The client code handles different service types properly. The client should be encode different types of payloads (int, float, string) based on the header's service type.
- **Payload Handling (10 points)**: Prints the header and the new payload returned by the server. The new payload is client header + payload. The header is the header sent by the sever.

Server Implementation (40 points)

- **Server Runs (10 points)**: The server code runs without errors.
- **Header handling (10 points)**: The server reads the header first, followed by the payload indicated by the header.
- **Decodes payloads properly**: The server code correctly decodes and prints the header and the payload.
- **Constructs and returns new payload =(10 points)**: The server creates a new payload from the header and payload and sends it back to client.

Best coding practices (20 points)

- **Error Handling (up to 10 points)**: Exit gracefully after client receives the message, Server never exits.
- **Code Clarity and Comments (up to 10 points)**: The code is well-commented and easy to understand, making use of meaningful variable and function names.

Total Points: /100